Constructing Computational Thinking Without Using Computers

Tim Bell, Michael Lodi

Constructing computational thinking without using computers[1]

Tim Bell • University of Canterbury, New Zealand • tim.bell/at/canterbury.ac.nz

Michael Lodi • University of Bologna & INRIA Focus, Italy • michael.lodi/at/unibo.it

Structured Abstract

Paper type: application.

Background(s): computer science; educational research.

Approach: Our approach is very practical: we are focused on pedagogy and improved classroom practices – what Matthews (1997: 8) calls "pedagogical constructivism." Moreover, we discuss the relationships between our work and Papert's constructionism.

Context: The meaning and implications of "computational thinking" (CT) are only now starting to be clarified, and the applications of the CS Unplugged approach are becoming clearer as research is appearing; now is a good time to consider how these relate, and what the opportunities and issues are for teachers using this approach.

Problem: The goal here is to connect computational thinking explicitly to the CS Unplugged pedagogical approach, and to identify the context where Unplugged can be used effectively.

Method: We take a theoretical approach, selecting a representative sample of CS Unplugged activities and mapping them to CT concepts.

Results: The CS Unplugged activities map well onto commonly accepted CT concepts, although caution must be taken not to regard CS Unplugged as being a complete approach to CT education.

Implications: There is evidence that CS Unplugged activities have a useful role to help students *and* teachers engage with CT, and to support hands-on activities with digital devices.

Constructivist content: A constructivist approach to teaching computer science concepts can be particularly valuable at present because the public (and many teachers who are likely to have to become engaged with the subject) don't see CS as something they are likely to understand; providing a clear way for anyone to

[1] This is an authors' pre-print version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in Constructivist Foundations 14, 3 (2019), 342–351. https://constructivist.info/14/3/342.bell

construct this knowledge for themselves provides an opportunity to empower them when it might otherwise have been regarded as a domain that is open to only a select few.

Key Words: computational thinking; CS Unplugged; Papert; teacher PLD; integrated learning; computation; algorithms; kinesthetic learning.

Introduction

1. Computer Science Unplugged (csunplugged.org) was originally intended as an outreach tool to explain computer science to young students, without the overhead of having to learn programming first. However, it is now used in a variety of contexts, and with the recent adoption of elements of computer science into school curricula around the world (Heintz et al. 2016, Webb et al. 2019), the Unplugged approach has often found a role in the classroom. The new curricula are commonly based around the idea of computational thinking, an idea that came to prominence after the publication of a paper by Jeannette Wing (2006), although the term was used by Papert as early as 1980 in his widely read "Mindstorms" book (Papert 1980), and the general concept predates Papert's work (Tedre and Denning 2016).

2. Here we look at what computational thinking is, how it relates to Computer Science Unplugged activities, and how this connects to previous research on the use of CS Unplugged for teachers and students.

3. There are many "unplugged" activities that aren't necessarily based on csunplugged.org (sometimes under titles such as kinesthetic activities), with some contributed by an international community of educators, but the key elements in the approaches we are exploring here are that computers aren't required despite all of the concepts being from the computer science canon, that students are engaged in kinesthetic activities, and any equipment needed is readily available at low cost (and would often be on hand in a classroom). The term "unplugged" is sometimes used to refer to the curated activities on the open-source CS Unplugged website (csunplugged.org), but in other contexts refers to any activity relating to computer science carried out away from a computer. In this paper we will use the full title ("CS Unplugged") to refer to the collection on the website[2], and "unplugged" when referring to the general concept of teaching computer science away from a computer.

4. In K-12 Computer Science education, constructivism (and especially constructionism) might normally be associated with computer programming. Other areas such as algorithm analysis, computability, formal languages, graphics and AI could be seen as theoretical knowledge that can be acquired as needed, potentially at a later stage. However, the CS Unplugged approach inverts a traditional "programming first" view by throwing students directly into advanced concepts in topics like graph theory, error correcting codes and computational complexity. Instead of taking a

[2] csunplugged.org

theoretical approach, students are usually given a kinesthetic experience in which they explore the issues in a way that is age-appropriate, and can engage with the ideas using a constructivist pedagogy. The purpose isn't primarily for students to acquire theoretical knowledge, but to appreciate the richness of the subject, and to give a meaningful experience those students who may not find programming to be engaging, but are interested in exploring some of the deeper issues that come up when they have access to computation through programming.

5. In this paper we explore the relationship (or rather, relationships) between computational thinking and the CS Unplugged material. We first consider what is meant by computational thinking, and then review the origins and intention of the "unplugged" approach. A central reason for linking them is identifying the computational thinking elements in a sample of CS Unplugged activities, and reflecting on how the activities can be applied effectively in practice.

Computational thinking

6. The term "computational thinking" (CT) is commonly used in the context of introducing computer science into primary and secondary school (K-12) education. A popular definition is the "Cuny-Snyder-Wing" formulation that dates from 2010:

> "Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent." (Wing 2010)

7. Being able to explain and implement CT in a K-12 context is important because it has been used as the basis of new curriculum material being introduced in several countries. The term "computational thinking" gained significant visibility through Jeannette Wing's writing in 2006 (Wing 2006), although the concepts pre-date that work significantly (Tedre and Denning 2016). There have been a variety of views about the full meaning and implications of the term "computational thinking", particularly around what an "information-processing agent" might be (Denning 2017), how much it hinges on the skill of computer programming (Curzon et al. 2019), and also any claims around the value to school students of learning CT (Tedre and Denning 2016). Curzon et al. (2019) point out that despite the diverse views about CT, there is considerable overlap, and it is best to focus on the points of agreement.

8. To understand CT, it is helpful to focus on what computation is. The notion of computation and computability has been explored in depth over the years, with fundamental ideas being based on Turing's work in the 1930s about what he called an "automatic machine" (Turing, 1937), which is now commonly referred to as a Turing machine. The limitations of a Turing machine still define the boundaries of digital computation today (with the possible exception of the new developments in quantum computing), so both the power and limitations of computing are reflected in any Turing-complete programming language, which includes many widely used educational languages such as Scratch and Python (Aho 2011). From this point of view, it could be argued that CT is centered around learning to use such languages to their full extent (Denning 2017), since these dictate exactly what can and can't be done in computation;

for example, Aho (2011) argues that computational thinking should be based on clearly defined models. It is reasonable to be concerned that analogies to computation (such as cooking recipes) are imperfect and might even teach against some of the deeper principles, but we need to acknowledge that beginners in any subject are often given simplified models to help scaffold their learning, such as the Rutherford-Bohr model of the atom, Newtonian physics, or the division of history into discrete periods. The important thing is for curriculum designers be clear on the usefulness of an inaccurate model if it is being used as a "stepping stone" (Duncan and Rivet 2013). Denning and Tedre (2019) explicitly note that computational thinking for a professional will be quite different to that for a beginner; beginners can "barely scratch the surface" of the richness and depth that is offered by computation. A spiral approach to curriculum is normal, and the challenge is to provide beginners with computationally meaningful encounters without the burden of having to fully understand the nuances of computation. Of course, if a teacher does not already have a clear understanding of a concept, then there is a risk that their teaching of the concept will lean too heavily on the analogy rather than their understanding of the concept itself.

9. Returning to the Cuny-Snyder-Wing definition, it opens the possibility that the information processing agent might be a human, but working within some constraints or scaffolding to ensure that the student experience is providing a valid foundation for working with more mechanically deterministic digital systems. It is also possible to enforce computationally authentic elements of computing without using a digital device. For example, the CS Unplugged resources (Bell et al. 2009, Bell, Rosamond and Casey 2012) have an activity on sorting algorithms that puts a simple rule in place that only two values can be compared at once using a balance scale, and the comparison is done by a third party so that there is no memory of previous comparisons, other than placement of the weights. This forces students to explore the same kinds of algorithms that a digital device would have to use when sorting by comparison (based on *if* statements and arrays), so it isn't just an analogy, but an alternative physical implementation of the kind of computation that is possible if one could program a digital device (although the restriction isn't strictly enforced, and a student might choose to stretch the "rules", for example, by comparing more than two items at a time). The unplugged approach has the advantage that students don't need to learn about programming first before engaging with the algorithm, although ultimately a computer program is needed to fully experience all of the limitations encountered when implementing the algorithm, as well as the benefit of the ability to reliably follow many instructions in a short time. It provides a constructivist environment that allows students to come up with their own algorithms for sorting, including evaluating how efficient an approach is, and comparing different approaches.

10. Despite different views about the purpose of using CT as a basis for curriculum, and how much it should be focused on physical digital devices, there is general agreement about the underlying skills for CT, whether it is to support a student learning to program, or is more broadly helping students to "think like a computer scientist" (Wing 2006, 2010). Lists of skills have been produced by national organizations supporting new curricula, including Computing at School in England (Csizmadia et al. 2015) and the Computer Science Teachers Association (CSTA 2011). Selby and

Woollard (2013) examine a number of such definitions, and argue that the most relevant and useful elements are abstraction, decomposition, algorithmic design, evaluation, and generalization. An element that is missing from their list, but commonly appears in others, is "logical thinking", which they argue is too broad and not well defined. This is a fair comment, since logic is fundamental to other disciplines too and also could be seen as subsumed by the other elements in CT. In this work we use the elements that they found most relevant, but have also included logical thinking in our analysis so that it can be related to other definitions, although we acknowledge that it isn't always considered to be a defining aspect of CT.

11. Denning (2017) warns against over-generalizing these underlying skills; for example, decomposition can be applied to many situations, such as breaking down a large (non-computer) project into components, but in a computational context such an activity has constraints imposed by the nature of a computing environment, as well as good practice (such as decomposing a large program into modules with meaningful functions). In the end, as Nardelli (2019) points out, the key reason for using the term "computational thinking" is pragmatic:

> "… we probably need the expression as an instrument, as a shorthand reference to a well-structured concept, but it might be dangerous to insist too much on it and to try to precisely characterize it. It should serve just as a brief explanation of why computer science ... is a novel and independent scientific subject and to argue for the need of teaching informatics in schools." (Nardelli 2019: 32)

12. Here we focus on the most widely used concepts, and applying them in the context of computer science, which helps us to connect the general ideas with their meaning in a computational context. They are used in the sections below to identify the connections between CS Unplugged and CT. The list of elements that will be used are based on Selby and Woollard's analysis (but with logical thinking included), and are briefly defined in Table 1. These definitions have been synthesized from a number of sources, including the papers on CT referenced above, and the "Exploring Computational Thinking" website.[3]

Abstraction	Identifying what the most important aspects of a problem are and hiding the other specific details that we don't need to focus on.
Decomposition	Breaking down problems into smaller, more manageable, parts, and then focusing on solving each of these smaller problems.
Algorithmic design	Creating step-by-step processes that solve a problem or complete a task.
Evaluation	Identifying the possible solutions to a problem and judging which is the best to use, if they will work in some situations but not others, and how they can be improved. In computing this can involve a range of criteria, including time and space used for computation, usability and correctness.

[3] https://edu.google.com/resources/programs/exploring-computational-thinking/

Generalization	Taking a solution (or part of a solution) to a problem and generalizing it so it can be applied to other similar problems and tasks.
(Logical thinking)	Trying to make sense of things by observing, collecting data, thinking about the facts you know, and then figuring things out based on what you already know.

Table 1: CT thought processes (based on Selby and Woollard 2013)

CS Unplugged

13. The Computer Science Unplugged resources originated from academics who had been asked to share what they did as a career with their children's peers, who at the time were around 5 or 6 years old (Bell, Rosamond and Casey 2012). Rather than talk *about* computer science, they chose to *do* computer science with the children, and from this point of view, CS Unplugged relates directly to helping students "think like a computer scientist" (Wing 2006, 2010). Ideas were taken from university courses – often advanced courses – and repackaged as physical activities where information such as graphs and binary digits were represented tangibly.

14. A simple example is the "parity" card trick, where a two-dimensional forward error correction code is introduced as a way for the presenter to somehow determine which card has been flipped over by a member of the audience. Students explore ideas for how the trick might be done, and once they discover the concept of parity, they can explore questions like whether or not two flipped cards can be identified, if it will still work with larger numbers of cards, whether a 3-dimensional version is better, and so on. Students are physically manipulating two-sided cards which (from a computer scientist's point of view) are binary digits, but for the student they need only consider their physical appearance – cards that are a different color on each side.

15. Another example is a game exploring routing and deadlock based on passing colored objects around, with the goal of getting the correct colors to the corresponding player. The processes required to solve this quickly end up requiring backtracking and logical arguments to achieve the group's goals.

16. It is important to be clear at this point that CS Unplugged isn't a curriculum, and isn't intended to replace the opportunity for students to write programs on digital devices, but it is an adjunct pedagogy to enable learners to become aware of bigger ideas in computing without having the overhead of learning to program first, and also to engage in big ideas through physical movement rather than expecting all computing classes to be sitting in front of a screen. CS Unplugged is also useful for communicating succinctly to students – and more significantly, teachers and education officials – that there is a depth to computation beyond stereotypes of "cutting code." In a modern classroom environment, the Unplugged approach is intended to be integrated with learning to program, and this can be more effective than spending all of the available time on programming alone (Hermans and Aivaloglou 2017). When Unplugged originated, classroom computers were either too rare for students to be likely to have

access to them, or the focus was on teaching students how to *use* the computer for standard productivity tasks rather than explore computational ideas with it. This situation has changed in many classrooms, and where digital devices are available, the CS Unplugged material can now be explicitly linked to programming through a "plugging it in" follow-up to the activities (Bell and Vahrenhold 2018).

17. The CS Unplugged approach does not usually spell out algorithms to students, but rather, a problem is given, and students explore potential algorithms for themselves. For small instances of a problem (such as converting a number to a 4-bit binary representation, finding the shortest path in a layout with only a few vertices, or searching for an item hidden under one of a few cups) an ad-hoc or brute-force approach may find the solution easily, but as the size of the problem increases students start to encounter the need for more efficient and rigorous approaches. When asked to search for a value hidden under one of 30 cups, students often switch from a sequential search to (an approximation of) binary search, and when converting numbers to a binary representation they may discover that a greedy approach gets results, but with other challenges (such as minimal spanning trees or sorting) they may only come to appreciate that a better algorithm is needed; and for NP-complete problems, not only do we not expect them to find a fast algorithm, but they end up grappling with the idea that no-one has (yet!) found a fast solution.

18. The main goal of taking a constructivist approach like this isn't that students learn particular algorithms and techniques, but that they learn that there are deep issues to be resolved in these contexts, and that they can feel empowered when they discover concepts for themselves, which can break stereotypes about what the qualities of a successful computer scientist might be.

CS Unplugged, constructivism and constructionism

19. Although the unplugged approach clearly differs from Papert's constructionism because the latter recognizes programming as having a leading role as a meta-tool for constructing knowledge, we can point out some relationships. First of all, unplugged activities are concrete rather than formal, and aim to teach complex CS ideas to children, ideas that are usually postponed until they become adult/formal/abstract thinkers. CS concepts are not simplified, but instead made accessible with practical experiences. Second, unplugged activities generally have children using their bodies or the physical manipulation of objects to perform them.

20. Similarly, Papert aimed to teach deep mathematical ideas long before children had the abstraction competence to grasp them formally: "My conjecture is that much of what we now see as too 'formal' or 'too mathematical' will be learned just as easily when children grow up in the computer-rich world of the very near future" (Papert 1980: 7). Moreover, he designed LOGO to be "body syntonic": children could use their body to impersonate the Turtle drawing on the screen: "working with the Turtle mobilizes the child's expertise and pleasure in motion. It draws on the child's well-established knowledge of 'body-geometry' as a starting point for the development of bridges into formal geometry" (Papert 1980: 58).

21. Thus, unplugged activities are a play space in which ideas from computer science can be explored, and the direction students take can be unpredictable. A constructivist — as opposed to instructionist (Papert 1993: 137-156) — approach is strongly advocated, as the primary goal is not for students to learn the concepts, but for them to discover that there are concepts that they may find interesting, and are worthy of study. Nevertheless, as computer science (and computational thinking) have started to enter school curricula, teachers have looked for ways to engage their students with specific ideas. The original activities were presented by CS researchers who were used to asking questions and exploring problems that may not have solutions, but in a classroom situation, teachers may not be experts, and aren't necessarily in a position to recognize the value of a direction a student might be taking an idea in. For this reason, the main version of CS Unplugged that is available gives considerable guidance on scaffolding the students' exploration of the ideas (Wood et al. 1976: 90), and provides questions for the teacher to ask, which can create a kind of Socratic method that enables students co-construct the meaning by following fruitful paths in their exploration (Wells 1999).

22. The degree of freedom left to students for exploration, and the role assumed by the teacher in guiding the activity (not as an expert delivering knowledge, but as a facilitator helping students experiment with ideas and constructing their own knowledge) plays, of course, a central role in the constructivist application of Unplugged material. This approach embraces the view that constructivism is a blend of more structured guidance and exploration, so it is not minimally guided, but *optimally* guided (Taber 2011), since the teacher has a path in mind, but nevertheless, the student is constructing the knowledge for themselves, and not having the ideas given to them directly.

Computational thinking and CS Unplugged

23. As pointed out in paragraph 13, CS Unplugged was intended to help children to understand what a computer scientist does, and CT has been referred to as "thinking like a computer scientist" (Wing 2006, 2010), matching Unplugged activities with CT ideas is useful to show how they are both serving a similar purpose. In fact, Wing's 2010 article specifically cites CS Unplugged under the heading of "Computational Thinking in Education."

24. Here we will show more explicit links using three contrasting activities: the binary representation activity (about data), searching (about algorithms), and sorting networks (applying a parallel algorithm to data). The CS Unplugged activities provide an environment that is intended to be used in a constructivist manner to scaffold learning, so that students are discovering patterns and rules for themselves based on a very short description of a challenge, rather than being told algorithms or solutions and then applying them. This means that they are exercising the CT skills themselves as they solve the challenges that they are given. The three activities were chosen as they span the range of approaches in CS Unplugged, including indoor vs. outdoor activities, working with and without given algorithms, and covering data and algorithms.

Figure 1: Representing 5-bit binary values using cards

25. In the binary representation activity, students manipulate cards that represent the powers of two (Figure 1). They follow the simple rule that the dots on a card may either be completely visible or not (depending on whether or not the card is flipped over), and are asked questions that lead them to find ways to display a given number of dots, counting, and exploring patterns in the representation. The constructivist approach means that the students are given little more instruction than the constraint that dots on a card are entirely visible or not (this is enforcing a constraint that a physical computing device would have); even the number of dots on each card should be deduced by the students after being shown the first three.

26. This activity is exercising the CT skills as follows; note that these observations could also be used to evaluate how well a student is applying the skills.

- Abstraction: Although binary representations are commonly said to be made of zeroes and ones, there are no such physical digits on a computer, only abstract representations. Students can experiment with a variety of abstractions; the activity starts by eliciting "yes" and "no" for the visibility of each card, but can then ask the students to be creative with other binary symbols, such as using two different musical pitches, dance moves, or even animal sounds. This then progresses to having them come up with more abstraction – binary symbols represent numbers, and then the numbers can represent other symbols, such as letters of the alphabet, months of the year, or colors of pixels in an image. And of course, the binary symbols are an abstraction for electrical signals; a feature of computer science is that it regularly deals with "multiple levels of abstraction" (Wing 2006).
- Decomposition: the problem of working out a number representation can be overwhelming at first, but students can use a left-to-right algorithm that decomposes it in simpler steps ("should this card be visible?"), and make it a lot simpler to comprehend. Another way of viewing this is that the concept of *number* is decomposed into a series of yes-no questions, where the first question (for 5-bit numbers) is "Is the value greater than or equal to 16?"
- Algorithmic design: Although this activity is about data, students are applying algorithms to the bits. Working out the representation of a decimal value can be done using the already-mentioned greedy algorithm working from left to right ("Should the 16-dot card be included?" etc.) Initially students may take a haphazard approach, but by scaffolding the idea of working from left to right, it becomes clear that the decisions can be easy to make. Other algorithms that come up are incrementing the displayed value by one (students can be scaffolded to discover that this can be done by flipping cards from right to left until a white card comes up),

doubling a value (shift left), and determining if a number is odd or even (simply check the right-hand bit!)

- Evaluation: Being able to convert between decimal and binary numbers isn't a widely used skill, but being able to evaluate the limits of a representation is. For example, students can evaluate the largest number possible with, say, 5 bits, and then with 6 bits, and with scaffolding, realize that each extra bit doubles the range of possibilities. This can lead to reasoning about the effectiveness of, say, a 256-bit security key vs. a 512-bit key; or an 8-bit character representation vs. 16-bit. In both cases the increase in representation is considerably more than the factor of 2 that might appear on the surface.
- Generalization: Starting from concrete examples, there are many patterns for students to explore here; the first generalization is working out the number of dots on the *n*-th card, but students can also discover many other patterns, for example, that the maximum value that can be represented with *k* bits is one less than the value of bit *k*+1, or that when counting, each card is being flipped with half the frequency of the one to its right. In doing this, students are moving from specific examples to general laws.
- Logical thinking: There are several rules that students can deduce using logical reasoning. A useful one is the uniqueness of a binary representation, based on the greedy algorithm used to find the representation. For example, suppose they have found the representation 01001 for the number 9. The students can then be asked "is it possible to have a representation of 9 where the first bit is 1?" They are likely to argue that it's not possible because you would have 16 dots – too many. The first bit must be 0. They then consider if the second bit could be a 0. Students will soon realize that there are only 7 dots left in that case, and can argue themselves that the second bit must be 1. This reasoning can be applied to all the bits of any binary representation, and students will have created an informal proof that a particular value has a unique representation.

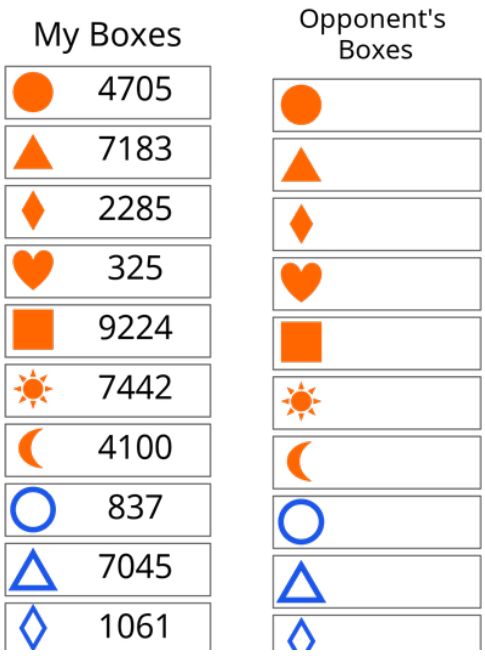

Figure 2: CS Unplugged "Number hunt" searching algorithms activity

27. In the searching algorithms activity, students are given hidden values to search in several contexts: for example, it could be hidden numbers held by a friend who will reveal the contents only one at a time (Figure 2), or cups that have values hidden under them that can only be revealed one at a time. In both cases, the goal is to find a value without looking at more items than necessary, and enforces the model of memory in which only one item can be looked up at a time. At first the values are unsorted, but students are later given a series of values that are sorted, which they can constructively use to apply a form of binary search to avoid frustratingly long searches.

28. Computational thinking appears in the searching activities as follows.

- Abstraction: The algorithms used apply to any kind of sorted and unsorted list, and depend only on the ability to impose an order on the keys. The hidden values themselves represent a sorted (or unsorted) list, and so has a property that isn't physically visible, but is crucial to the algorithm used for the searching. The algorithm abstracts from the actual numbers in a list; for example, in binary search, the concrete values on the cards to the left of the midpoint may not be known, but the student can know that the key value isn't on any of them.
- Decomposition: Each key comparison is decomposing the solution space into smaller parts; in the case of a sequential search it is a small gain (the solution space is reduced one by one), but students can explore the power of divide-and-conquer through binary search, where a half of the problem space is eliminated in one step.
- Algorithmic design: Students constructively discover a (variation of) the binary search algorithm motivated by minimizing the cost to them of finding a given object or value. They can also recognize a sequential search will work on any list – including a sorted list!
- Evaluation: This is a key reason for students to explore searching algorithms; sequential and binary search have quite different performance, and although students may not use mathematical language like "logarithms," they can appreciate that even billions of items can be searched in a very small time with binary search, and that sequential search becomes arbitrarily worse than binary search as the size of the list increases. Initially when a teacher suggests performing a binary search on twice as many objects, students can appear quite disheartened, until they evaluate the algorithm and realize that only one extra step is required.
- Generalization: The different guises of searching (numbers on cards, cups, envelopes and so on) are all the same problem with the same possible solutions, but presented in different ways. This enables students to recognize what the general algorithm is, rather than just a specific application of it. The algorithm can also be applied to text and dates; students should recognize that binary search can work with any keys as long as they can be sorted into order (i.e. are from a binary relation that is a total order).
- Logical thinking: There are a number of ideas that can be reasoned about here: for example, that binary search can only work on a sorted list, and that binary search is guaranteed to find what the student is looking for even though many items are never inspected.

Figure 3. The CS Unplugged sorting network activity

29. In the sorting network activity, students traverse a network drawn on the ground, making a simple comparison of values at each node and taking the left or right exit based on the comparison (Figure 3). At the end they discover that the values have been sorted into ascending order. Again, the instructions given are very simple (compare values and go left or right), but the activity allows the students to construct a range of understandings based on their experience, and engage with CT concepts.

- Abstraction: The sorting network is a physical representation of what happens inside a computer. The way the network is drawn (e.g. large or small boxes, long paths, and paths that take roundabout-routes) doesn't matter as long as the topology of the network is maintained. The values being compared (keys) are also an abstraction of some item that is being sorted, which may have more data than just the sort key.
- Decomposition: In this activity students aren't *doing* decomposition, but are experiencing the result of it. A key aspect of this activity is that the complex task is decomposed into a very simply described task; in a physical computer this relates to the most complex algorithms being performed by combinations of just a few simple instructions. In this case, students are often surprised that such simple instructions result in a powerful outcome. When comparing words or large numbers, these are further decomposed into a character-by-character or digit-by-digit comparison to determine which value comes first.
- Algorithmic design: The students are physically engaging with a parallel algorithm, and seeing how a complex outcome (sorting) can be achieved by the combination of many simple steps (in this case, comparisons of pairs of values). They also have the opportunity to design their own parallel algorithms for smaller sorting networks. If there is an opportunity to dig deeper, students are able to realize that the parallel algorithm can in turn be designed by an algorithm, which raises some interesting philosophical issues about algorithm design!
- Evaluation: A sorting network can be evaluated in terms of the number of nodes required (three at a time in the case of a 6-way network), but also in terms of the number of parallel steps required (the length of the network, and therefore time required). Two different networks for the same number of inputs can be compared based on these metrics.

- Generalization: The sorting network can be used to sort any values that form a binary relation that is a total order (i.e. values that can be consistently compared for inequality); numbers are in order of increasing value, while words are in alphabetical order, but the comparisons can also be used for other types of data, such as musical notes (higher and lower), and stories (which plot element comes before another?)
- Logical thinking: Students are able to reason about the correctness of the configuration by applying logic (an exhaustive test would take $n!$ time). A first step is to apply logic to reason that the smallest item must end up in the correct place, regardless of where it starts. A full proof of correctness is likely beyond students, but for small sorting networks there is a lot of opportunity to reason about what will happen.

30. These activities have been used as examples; on the CS Unplugged website CT skills are made explicit for every activity to help teachers see the bigger picture of why a particular activity is relevant, to help them appreciate which finer details of an activity are important to fully engage students in CT, and to support them to recognize when a student is showing CT skills. The descriptions on the website have been informed by the reasoning above.

Integrated multidisciplinary learning

31. Although computing may be taught as an independent subject, it makes more sense when it is used in context. In the same way, computational thinking (and computer science) isn't an end in itself, and is applied in many practical contexts. When taught in schools it can be used effectively in multidisciplinary contexts, where CT concepts are applied in other subjects to support learning in both at the same time. With "plugged in" approaches this may be easier to see, as students can write programs to simulate situations they are learning about, to capture and analyze scientific data, to generate music and artwork, to make sense of health information captured by personal fitness devices, and so on. However, even with an unplugged approach there are many possibilities for integrated learning.

32. For example, the binary representation activity includes the possibility of threading beads chosen from two colors into bracelets, necklaces or bag-tags. Making up chains of beads gives students the chance to think about language, and what they would like to communicate with the beads; it also happens to exercise fine motor skills. More generally, the activity can be extended to art, where two symbols or images are used to embed information in a picture; or music, where the two values can be used as note pitches or lengths. Both of these can be used to introduce the idea of steganography, where a message is communicated in plain sight through an artefact that appears to have a different purpose. Topics like binary numbers can also be integrated with history and writing – where did the idea come from, and how have people communicated in the past over distance? Looking into representations like Braille and Morse code can reveal how communication has influenced history, but also how it is natural for humans to develop codes for communication over distance or for efficient storage. Students can construct their own codes based on their new understanding, and this provides a richer experience

than simply learning standard codes (such as ASCII and Unicode), as they will face the questions that arise for themselves, such as special characters, using digits as text, and so on.

33. Teachers may not think to connect a "modern" subject like computation with traditional subjects such as history. However, there are a number of ways this can be done, and articulating a few examples can inspire teachers to find their own. Sorting networks can be integrated with topics that students are exploring in other areas of the curriculum; for example, they might be used to compare dates in history, words in alphabetical order, note pitches in music, or numbers written in a foreign language. They provide motivation for students to repeatedly compare the values that they are learning, and to see them in situations other than the sequence normally presented. At the same time, they are becoming familiar with a computational model.

34. Searching algorithms can also be explored in terms of history – how did people look up information in pre-computer times, and who had access to such information? Who are the people who developed these computer algorithms, and what motivated them? There is also the possibility of acting out such algorithms; and a binary search can even be used to compare an unknown pitch with the notes on the piano to determine what it is.

35. It is challenging, in a traditional school setting, to always propose activities that are interesting and meaningful for every student. However, connection with real-world applications of CS (e.g. understanding what's behind a query in a search engine, or how data is physically transmitted online, and so on) helps to create engagement and meaning, because it links to previous knowledge, and personal and social experiences of students, which are essential in constructivist learning.

Applying CS Unplugged

36. An important feature of this style of teaching is to give minimal instructions (often just one or two sentences are sufficient to get students started), and allow students to construct the knowledge for themselves. Once they have done this, it is important to then relate what they have done to the broader context of computing, and what happens on physical devices. Two early studies discovered that without this connection "the program [based on CS Unplugged] had no statistically significant impact on student attitudes toward computer science or perceived content understanding" (Feaster et al. 2011) and that "the students' attitudes and intentions regarding CS did not change in the desired direction" (Taub, Armoni and Ben-Ari 2012). In terms of conveying knowledge using this approach compared with more conventional approaches, Thies and Vahrenhold (2013) found that "… it is indeed possible to weave Computer Science Unplugged activities into lower secondary computer science classes without a negative effect on factual, procedural, or conceptual knowledge", and that it could have some benefit in that "the Computer Science Unplugged materials can prove helpful for ability grouping within a class, since, on average, more students are enabled to reach a higher operational stage."

37. Gains from using an unplugged approach were reported by Hermans and Aivaloglou (2017), who combined it with teaching programming for one group, while having a second group spend the same total amount of time learning only programming; they found that "…the group taught using CS Unplugged material showed higher self-efficacy and used a wider vocabulary of Scratch blocks."

38. Looking at these different contexts, we see that CS Unplugged is best used in combination with "plugged in" work. This is not surprising, given that getting a program to work correctly is an excellent way for a student to show that they have understood the computational concepts they are working with, since the computational agent (the computer running the program) will do exactly what the program says to do. Moreover, this will give students the opportunity to experience in a tangible (in some sense) environment the effects of their instructions, with immediate and unexceptionable feedback (rather than delayed feedback from another person, typically the teacher). Based on this, the CS Unplugged website now offers a range of "Plugging it in" exercises to provide follow-up activities that allow students to link their unplugged learning with computation on a digital device.

39. An unplugged approach seems to have promise for helping student learning if used effectively, but another important value of it is for teachers. Teachers need to be confident in a topic so that they can build student confidence, and given that the new computing curricula appearing around the world are often taught by people new to the subject, ways to build teacher confidence will be important (Gutiérrez and Sanders 2009). Often teachers are intimidated by new terminology – words such as "algorithm" and "binary" appear in curricula, but looking up definitions of such terms often results in a description that is meaningless to the layperson, whereas the CS Unplugged material gives an opportunity to engage with the concept, and then learn what its name is, which is a much more meaningful way to learn new terminology.

40. CS Unplugged has been used in a variety of teacher professional learning and development (PLD) initiatives, and the research available on this is reporting positive outcomes. For example, Curzon et al. (2014) report on teacher professional development that had a substantial "unplugged" component, and noted that it was "inspiring, confidence building and gave [the teachers] a greater understanding of the concepts involved." An important feature of the constructivist approach of Unplugged activities is that they allow very quick wins, where teachers can understand a new concept (such as binary numbers) very quickly, in the context of a hands-on first-person experience, without the overhead of having to learn to program first. Smith et al. (2015) reported that teachers who were training other teachers (through the UK Master teachers system) commonly included CS Unplugged when providing professional development for colleagues, and both Morreale and Joiner (2011) and Sentance and Csizmadia (2017) found that after attending their workshop, CS Unplugged was widely adopted by teachers.

Conclusion

41. CS Unplugged activities can provide scaffolding to support a constructivist approach to introduce computer science without computers, helping students construct their own initial knowledge about key ideas behind deep computational thinking concepts through kinesthetic experiences. Given the genesis of the CS Unplugged material, it isn't surprising that it strongly engages students in computational thinking, and the examples above illustrate how both CS Unplugged and CT (which have independent origins) are aligned. These links can also be used in the classroom to identify when students are exhibiting CT skills.

42. Nevertheless, to be effective, unplugged approaches should be used thoughtfully – any pedagogical approach can be delivered badly with little effort! The constructivist character of the activities needs to be maintained, and we know that unplugged activities are effective when used in a context where they will be ultimately linked to implementation on a digital device, either through programming, or by helping students to see where these ideas impinge on their daily life. By using CS Unplugged early to introduce concepts, both students and teachers new to the subject can have early success without the overhead of becoming proficient enough at programming to engage properly with ideas that can have an impact in our digital world. This then provides a useful platform to motivate learning the skill of programming, but also a way to connect computer science with other subjects.

References

Aho, A. V. (2011). What is Computation? Computation and computational thinking. *Ubiquity Symposium*, (January), 1–10.

Bell, T., Alexander, J., Freeman, I., and Grimley, M. (2009). Computer science unplugged: school students doing real Computing without computers. *New Zealand Journal of Applied Computing and Information Technology*, 13(1), 20-29.

Bell, T., Rosamond, F., and Casey, N. (2012). Computer Science Unplugged and related projects in math and computer science popularization. In H. L. Bodlaender, R. Downey, F. V Fomin, and D. Marx (Eds.), *The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, LNCS 7370, pp. 398–456. Springer-Verlag.

Bell, T., and Vahrenhold, J. (2018). CS Unplugged—How Is It Used, and Does It Work? In *Adventures Between Lower Bounds and Higher Altitudes* (pp. 497–521). Springer.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., and Woollard, J. (2015). Computational thinking: a guide for teachers. Available from http://computingatschool.org.uk/computationalthinking.

CSTA. Operational Definition of Computational Thinking. 2011; https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CompThinkingFlyer.pdf

Curzon, P., Bell, T., Waite, J., and Dorling, M. (2019). Computational thinking. In S. Fincher and A. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 513–546). Cambridge University Press.

Curzon, P., McOwan, P. W., Plant, N., and Meagher, L. R. (2014). Introducing teachers to computational thinking using unplugged storytelling. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education – WiPSCE '14*, 89–92.

Denning, P. (2017) Remaining trouble spots with computational thinking, *Communications of the ACM*. 60(6): 33-39, June.

Denning, P., and Tedre, M. (2019). *Computational Thinking*. Boston, MA: MIT Press Essential Knowledge series.

Duncan, R. G., and Rivet, A. E. (2013). Science learning progressions. *Science*, *339*(6118), 396-397.

Feaster, Y., Segars, L., Wahba, S. K., and Hallstrom, J. O. (2011). Teaching CS unplugged in the high school (with limited success). In G. Rößling, T. L. Naps, and C. Spannagel (Eds.), *Proceedings of the 16th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011*, Darmstadt, Germany, June 27-29, 2011 (pp. 248–252). ACM.

Gutiérrez, J. M., and Sanders, I. D. (2009). Computer Science education in Perú: a new kind of monster? *ACM SIGCSE Bulletin*, *41*(2), 86–89.

Heintz, F., Mannila, L., and Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K-12 education. In Proceedings – Frontiers in Education Conference (FIE) (pp. 1–9).

Hermans, F., and Aivaloglou, E. (2017). To Scratch or Not to Scratch?: A Controlled Experiment Comparing Plugged First and Unplugged First Programming Lessons. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (pp. 49–56). New York, NY, USA: ACM. http://doi.org/10.1145/3137065.3137072

Matthews, M. R. (1997). Introductory comments on philosophy and constructivism in science education. *Science & Education*, 6(1-2), 5-14.

Morreale, P., and Joiner, D. (2011). Reaching future computer scientists. *Communications of the ACM*, *54*(4), 121.

Nardelli, E. (2019). Do we really need computational thinking? *Communications of the ACM*, *62*(2), 32–35. http://doi.org/10.1145/3231587

Papert, S. (1980) Mindstorms: Children, Computers and Powerful ideas. Basic Books.

Papert, S. (1993). The Children's Machine: Rethinking School in the Age of the Computer. Basic Books.

Selby, C. C., and Woollard, J. (2013). Computational Thinking: The Developing Definition. University of Southampton (E-prints) 6pp. https://eprints.soton.ac.uk/356481/

Sentance, S., and Csizmadia, A. (2017). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies*, *22*(2), 469–495. http://doi.org/10.1007/s10639-016-9482-0

Smith, N., Allsop, Y., Caldwell, H., Hill, D., Dimitriadi, Y., and Csizmadia, A. P. (2015). Master teachers in computing: What have we achieved? In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 21–24).

Taber, K. S. (2011). Constructivism as educational theory: Contingency in learning, and optimally guided instruction. In J. Hassaskhah (Ed.), Educational Theory. Hauppauge, NY: Nova Science Publishers, Inc., pp. 39-61.

Taub, R., Armoni, M., and Ben-Ari, M. (2012). CS Unplugged and Middle-School Students' Views, Attitudes, and Intentions Regarding CS. *Trans. Comput. Educ.*, *12*(2), 8:1–8:29. http://doi.org/10.1145/2160547.2160551

Tedre, M., and Denning, P. J. (2016). The Long Quest for Computational Thinking. In *Proceedings of the 16th Koli Calling Conference on Computing Education Research*, pp. 120–129.

Thies, R., and Vahrenhold, J. (2013). On Plugging "Unplugged'' into CS Classes. In *SIGCSE '13: Proceedings of the 44th ACM technical symposium on Computer Science Education* (pp. 365–370).

Turing, A. M. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1), 230-265.

Webb, M., Bottino, R. M., Passey, D., Kalas, I., Bescherer, C., Smith, J. M., … Fuschek, G. (2019). Coding , Programming and the Changing Curriculum for Computing in Schools: Report of UNESCO/IFIP TC3 Meeting at OCCE – Wednesday 27th of June 2018, Linz, Austria.

Wells, G. (1999). Dialogic Inquiry: Towards a Sociocultural Practice and Theory of Education. New York: Cambridge University Press

Wing, J.M., (2006) Computational Thinking, *Communications of the ACM*. 49 (3)

Wing, J.M. (2010). Computational Thinking: What and Why? *The Link Magazine (Carnegie Mellon University)*, Spring. Retrieved from https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wood, D., Bruner, J., and Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Child Psychiatry*, 17, 89−100.

The authors

Tim Bell is a full professor in the Department of Computer Science and Software Engineering at the University of Canterbury in Christchurch, New Zealand. His main current research interest is computer science education; in the past he has been also worked on computers and music, and data compression. His "Computer Science Unplugged" project is widely used internationally, and its books and videos have been translated into about 25 languages. He has received several awards for his work in CS communication, including the 2018 ACM SIGCSE Outstanding Contribution to Computer Science Education award, and he is an ACM Distinguished Member. He has co-authored four books, and around 140 academic papers.

Michael Lodi is a PhD student in Computer Science, in the Department of Computer Science and Engineering, University of Bologna, Italy. He has also received a BS, MS and High school teaching license in CS from the same University. He works on computer science education, with a particular focus on teacher training about computational thinking and epistemological aspects of Computer Science as a discipline. In particular, he studies "Computer Science Growth Mindset". He has published some papers in international conferences on computer science education,

and a book in Italian for primary school teachers. He is actively involved in nationwide initiatives to introduce CS in Italian K-12 curriculum. https://lodi.ml